AMENDMENTS TO THE CLAIMS

Please replace the claims as follows:

1. (previously presented, last amended 12/30/05) A method, comprising:

without modifying a pre-existing operating system of the computer, establishing an entry exception to be raised on entry to the operating system at a specified entry point or on a specified condition, the entry exception having an associated entry handler, the entry handler programmed to save a context of an interrupted thread and modify the thread context before delivering the modified thread context to the operating system;

without modifying the operating system, establishing a resumption exception to be raised on resumption from the operating system when such resumption is complementary to one of the specified entries, the resumption exception having an associated exit handler, the exit handler programmed to restore the context saved by a corresponding execution of the entry handler;

scheduling concurrent threads of control by the operating system, each thread having an associated context, the association between a thread and a set of computer resources of the associated context being maintained by the operating system;

on detecting a specified entry to the operating system from an interrupted thread of the computer, raising and servicing the entry exception; and

on detecting a complementary resumption, raising and servicing the resumption exception, and returning control to the interrupted thread;

the entry exception, resumption exception, entry handler, and exit handler being cooperatively designed to maintain an association between one of the threads and an extended context of the thread through a context change induced by the operating system, the extended context including resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the operating system.

2. (original) The method of claim 1, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer.

3. (original) The method of claim 1, wherein the operating-system-maintained resources of the thread context include data registers of the non-native computer architecture, the method further comprising:

modifying at least half of the data registers of the portion of the thread context maintained by the operating system before delivering the thread to the non-native operating system.

4. (original) The method of claim 1, wherein thread scheduler and the thread execute in different instruction sets of the computer, and the entry and exit exception are automatically invoked, without explicit software request, on a transition between the thread instruction set and the operating system instruction set.

5. (previously presented, last amended 9/13/04) A method, comprising:

scheduling concurrent threads of control by a pre-existing thread scheduler of a computer, each thread having an associated context, an association between a thread and a set

of computer resources of the associated context being maintained by the thread scheduler; and

without modifying the thread scheduler, maintaining an association between one of the threads and an extended context of the thread through a context change induced by the thread scheduler, the extended context including resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the thread scheduler.

6. (previously presented, last amended 12/30/05) The method of claim 5, wherein the thread scheduler is a component of an operating system of the computer, and further comprising:

establishing an entry exception to be raised on entry to the operating system at a specified entry point or on a specified condition;

establishing a resumption exception to be raised on a resumption from the operating system following on a specified entry;

on detecting a specified entry to the operating system from an interrupted process of the computer, raising the entry exception, and establishing the association as part of servicing the entry exception; and

raising the resumption exception, and as part of servicing the resumption exception, reestablishing the context in association with the resumed thread returning control to the interrupted process.

7. (previously presented, last amended 12/30/05) The method of claim 6, wherein an exception handler for the entry exception is programmed to save a context of the interrupted process and modify the thread context before delivering the modified thread context to the operating system; and

an exception handler for the resumption exception is programmed to restore the context saved by a corresponding execution of the entry exception handler.

8. (original) The method of claim 7, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer.

9. (original) The method of claim 8, wherein the computer additionally executes an operating system native to the computer, and each exception is classified for handling by one of the two operating systems.

10. (original) The method of claim 8, wherein operating system and the interrupted thread execute in different instruction set architectures of the computer.

11. (original) The method of claim 6, wherein the operating system is in a binary code for a computer architecture non-native to the architecture of the computer.

12. (original) The method of claim 11, wherein the computer additionally executes an operating system native to the computer, and each exception is classified for handling by one of the two operating systems.

13. (original) The method of claim 11, wherein operating system and the interrupted thread execute in different instruction set architectures of the computer.

14. (original) The method of claim 11, wherein the resources of the context maintained in association with the thread by the non-native operating system include data registers of the non-native computer architecture, the method further comprising:

in the entry exception handler, modifying at least half of the data registers of the portion of the process context maintained by the non-native operating system before delivering the process to the non-native operating system, at least some of the modified registers being redundantly written with data to enable checking of the validity of the contents of the context in the resumption exception handler.

15. (original) The method of claim 6, wherein thread scheduler and the thread execute in different execution modes of the computer, and the steps to maintain the association between the thread and the context are automatically invoked, without explicit software request, on a transition between the thread execution mode and the thread scheduler execution mode.

16. (original) The method of claim 15, wherein the thread execution mode and the thread scheduler execution mode are two different instruction set architectures of the computer.

17. (original) The method of claim 6, further comprising:

during servicing the entry exception, saving a portion of the context of the computer, and altering the context of an interrupted thread before delivering the interrupted thread and its corresponding context to the operating system.

18. (original) The method of claim 6, further comprising the step of modifying a linkage return address for resumption of the thread to include information used to maintain the association.

19. (original) The method of claim 18, wherein the modification leaves at least half of the bits of the linkage return address intact.

20. (original) The method of claim 5, wherein the thread scheduler is an operating system for a computer architecture other than the architecture native to the computer.

21. (original) The method of claim 20, wherein the computer additionally executes an operating system native to the computer, and each exception is classified for handling by one of the two operating systems.

22. (original) The method of claim 20, wherein operating system and the interrupted thread execute in different instruction set architectures of the computer.

23. (original) The method of claim 5, wherein thread scheduler and the thread execute in different execution modes of the computer, and the steps to maintain the

association between the thread and the context are automatically invoked, without explicit software request, on a transition between the thread execution mode and the thread scheduler execution mode.

24. (original) The method of claim 23, wherein the thread execution mode and the thread scheduler execution mode are two different instruction set architectures of the computer.

25. (original) The method of claim 23, further comprising the step of setting of a register to a value that specifies actions to be taken by an exception handler invoked on the transition to convert operands from one form to another to conform to a data storage convention of the thread scheduler execution mode.

26. (original) The method of claim 5, further comprising:

in an interrupt handler of the computer, saving a portion of the context of the computer, and altering the context of an interrupted thread before delivering the interrupted thread and its corresponding context to the thread scheduler.

27. (original) The method of claim 20, wherein the operating-system-maintained resources of the thread context include data registers of the non-native computer architecture, the method further comprising:

modifying at least half of the data registers of the portion of the thread context maintained by the operating system before delivering the thread to the non-native operating system.

28. (original) The method of claim 27, wherein at least some of the modified registers are overwritten by a timestamp.

29. (original) The method of claim 27, wherein at least some of the modified registers are overwritten by information indicating a storage location at which at least the portion of the thread context to be modified is saved before the modifying.

30. (original) The method of claim 5, further comprising the step of modifying a linkage return address for the thread to include information used to maintain the association.

31. (original) The method of claim 30, wherein the modification leaves at least half of the bits of the linkage return address intact.

32. (previously presented, last amended 9/13/04) The method of claim 5, further comprising either the step of deferring delivery of an interrupt before interrupting the thread by a time sufficient to allow the thread to reach a checkpoint, or the step of rolling execution of the thread back to a checkpoint, the checkpoints being points in the execution of the thread where the amount of extended context, being the resources of the thread that are beyond those whose resource association with the thread is maintained by the thread scheduler, is reduced.

33. (previously presented, last amended 12/30/05) A method, comprising:

establishing an entry exception to be raised on entry to a computer operating system at a specified entry point or on a specified condition;

establishing a resumption exception to be raised on resumption from the operating system when such resumption is complementary to one of the specified entries;

on detecting a specified entry to the operating system from interruption of a process executing on the computer, raising and servicing the entry exception, and then entering the operating system to perform a service associated with the specified operating system entry; and

on detecting a complementary resumption, raising and servicing the resumption exception, and returning control to the interrupted process.

34. (previously presented, last amended 12/30/05) The method of claim 33, wherein an exception handler for the entry exception is programmed to save a context of the interrupted process and modify the process context before delivering the modified process context to the operating system; and

an exception handler for the resumption exception is programmed to restore the context saved by a corresponding execution of the entry exception handler.

35. (original) The method of claim 34, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer.

36. (previously presented, last amended 12/30/05) The method of claim 35, wherein operating system and the interrupted process execute in different instruction set architectures of the computer.

37. (previously presented, last amended 12/30/05) The method of claim 35, wherein the resources of the context maintained in association with the process by the operating system include data registers of the non-native computer architecture, the method further comprising:

in the entry exception handler, modifying at least half of the data registers of the portion of the process context maintained by the operating system before delivering the process to the non-native operating system, at least some of the modified registers being redundantly written with data to enable checking of the validity of the contents of the context in the resumption exception handler.

38. (original) The method of claim 34, wherein the operating system and the process execute in two different instruction set architectures of the computer, and at least some of the steps to maintain the association between the process and the context are automatically invoked, without explicit software request, on a transition between the instruction set architectures.

39. (original) The method of claim 34, further comprising the step of modifying a linkage return address for the process to include information used to restore the context.

40. (original) The method of claim 33, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer, unmodified for execution on the computer.

41. (previously presented, last amended 12/30/05) The method of claim 40, wherein the computer additionally executes an operating system native to the computer, and each exception is classified for handling by one of the native operating system or the operating system not native to the computer's architecture.

42. (previously presented, last amended 12/30/05) The method of claim 40, wherein operating system and the interrupted process execute in different instruction set architectures of the computer.

43. (previously presented, last amended 12/30/05) The method of claim 33, wherein the operating system and the interrupted process execute in different execution modes of the computer, and the steps to maintain the association between the interrupted process and the context are automatically invoked, without explicit software request, on a transition between the execution mode of the interrupted process and the execution mode of the operating system.

44. (original) The method of claim 43, wherein the process execution mode and the operating system execution mode are two different instruction set architectures of the computer.

45. (original) The method of claim 33, wherein a service routine for the entry exception modifies at least half of the data registers of the portion of a process context maintained in association with the process by the operating system before delivering the process to the non-native operating system.

46. (previously presented, last amended 9/13/04) The method of claim 45, wherein at least some of the modified registers are overwritten by information indicating a storage location at which at least the extended context, the extended context being the resources beyond those whose resource association with the process is maintained by the operating system, is saved before the modifying.

47. (previously presented, last amended 12/30/05) The method of claim 46, wherein at least some of the modified registers are overwritten by a value that enables validation of the contents of the process or extended context.

48. (original) The method of claim 45, wherein at least some of the modified registers are overwritten by a value that enables validation of the contents of the context.

49. (original) The method of claim 45, wherein at least some of the modified registers are overwritten by a timestamp.

50. (original) The method of claim 33, further comprising the step of modifying a linkage return address for the process to include information used to maintain the association.

51. (original) The method of claim 50, wherein the modification leaves at least half of the bits of the linkage return address intact.

52. (original) The method of claim 33, further comprising:

as part of servicing the entry exception, modifying a linkage return address of the interrupted process, the return address being deliberately chosen so that an attempt to execute an instruction from the return address on return from the operating system will raise the resumption exception.

53. (original) The method of claim 52, wherein the linkage return address is selected to point to a memory page having a memory attribute that raises the chosen exception on at attempt to execute an instruction from the page.

54. (previously presented) The method of claim 33, further comprising either the step of rolling execution of the process back to a checkpoint in the execution of the process where the amount of extended context, the extended context being the resources of the process context beyond those whose resource association with the process is maintained by the operating system, is reduced.

55. (previously presented) The method of claim 33, further comprising either the step of deferring delivery of an interrupt before interrupting the process by a time sufficient to allow the process to reach a checkpoint in the execution of the process where the amount of extended context, the extended context being the resources of the process context beyond those whose resource association with the process is maintained by the operating system, is reduced.

56. (previously presented, last amended 12/30/05) A method, comprising:

without modifying a pre-existing operating system of the computer, establishing an entry handler for execution at a specified entry point or on a specified entry condition to the operating system, the entry handler programmed to save a context of an interrupted thread and modify the thread context before delivering the modified thread context to the operating system;

without modifying the operating system, establishing an exit handler for execution on resumption from the operating system following an entry through the entry handler, the exit handler programmed to restore the context saved by a corresponding execution of the entry handler.

57. (previously presented) The method of claim 56, further comprising:

scheduling concurrent threads of control by the operating system, each thread having an associated context, an association between a thread and a set of computer resources of the associated context being maintained by the operating system; and

the entry and exit handlers being programmed to maintain an association between one of the threads and an extended context of the thread through a context change induced by the operating system, the extended context including resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the operating system.

58. (original) The method of claim 57, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer.

59. (original) The method of claim 57, wherein the operating system and the thread execute in different execution modes of the computer, and the steps to maintain the association between the thread and the context are automatically invoked, without explicit software request, on a transition between the thread execution mode and the operating system execution mode.

60. (original) The method of claim 57, further comprising:

in the entry handler, saving a portion of the context of the computer, and altering the context of the interrupted thread before delivering the interrupted thread and its corresponding context to the operating system.

61. (original) The method of claim 57, wherein the entry handler alters at least half of the data registers of the portion of a thread context maintained in association with the thread by the operating system before delivering the thread to the operating system.

62. (original) The method of claim 57, further comprising the step of modifying a linkage return address for the thread to include information used to maintain the association.

63. (original) The method of claim 56, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer.

64. (original) The method of claim 63, wherein the computer additionally executes an operating system native to the computer, and each interrupt or exception is classified for handling by one of the two operating systems.

65. (original) The method of claim 63, wherein operating system and the interrupted thread execute in different instruction set architectures of the computer.

66. (original) The method of claim 56, wherein the operating system and the thread execute in different execution modes of the computer, and the steps to maintain the association between the thread and the context are automatically invoked, without explicit software request, on a transition between the thread execution mode and the operating system execution mode.

67. (original) The method of claim 66, wherein the thread execution mode and the operating system execution mode are two different instruction set architectures of the computer.

68. (original) The method of claim 56, wherein the operating system maintains an association between contexts and corresponding threads of execution, each such context including values of data registers, the method further comprising:

modifying at least half of the data registers of the portion of the thread context maintained by the operating system before delivering the thread to the operating system.

69. (original) The method of claim 68, wherein at least some of the modified registers are overwritten by information indicating a storage location at which at least the portion of the thread context to be modified is saved before the modifying.

70. (original) The method of claim 68, wherein at least some of the modified registers are overwritten by a value that enables validation of the contents of the context.

71. (original) The method of claim 56, further comprising the step of modifying a linkage return address for the thread to include information used to restore the context of the thread.

72. (previously presented, last amended 12/30/05) The method of claim 71, wherein the linkage return address is modified with information indicating an execution path by which, or a condition on which, execution arrived at the entry handler.

73. (original) The method of claim 71, wherein the modification leaves at least half of the bits of the linkage return address intact.

74. (previously presented, last amended 12/30/05) The method of claim 71, wherein the linkage return address is modified with information indicating a storage location at which at least the portion of the thread context to be modified is saved before the modifying.

75. (previously presented) The method of claim 56:

wherein the interrupted thread at the point of interruption executes in one instruction set architecture and the operating system is coded primarily in a different instruction set architecture; and

further comprising the step of setting of a register to a value that specifies actions to be taken by the entry handler or exit handler to convert operands from one form to another to conform to a data storage convention of the operating system instruction set architecture.

76. (original) The method of claim 56, further comprising either the step of deferring delivery of an interrupt before interrupting the thread by a time sufficient to allow the thread to reach a checkpoint in the execution of the thread where the amount of extended context, being the resources of the thread context beyond those whose resource association with the thread is maintained by the operating system, is reduced.

77. (original) The method of claim 56, further comprising either the step of rolling execution of the thread back to a checkpoint in the execution of the thread where the amount of extended context, being the resources of the thread context beyond those whose resource association with the thread is maintained by the operating system, is reduced.

78. (previously presented, last amended 12/30/05) The method of claim 56, further comprising either the step of storing at least the extended context, the extended context being the resources beyond those whose resource association with the thread is maintained by the operating system, into a storage location, a pool of storage locations being managed by a queuing discipline in which empty storage locations in which a context is to be saved are allocated from the head of a queue, recently-emptied storage locations for reuse are enqueued at the head of the queue, and full storage locations to be saved are queued at the tail of the queue.

79. (original) A method, comprising:

during invocation of a service routine of a computer, passing a linkage return address to the service routine at which to resume execution on completion of the service, the linkage return address being deliberately chosen so that an attempt to execute an instruction from the linkage return address on return from the service routine will raise a program execution exception;

on return from the service routine, attempting to execute the instruction at the linkage return address and raising the chosen exception; and

after servicing the exception, returning control to a caller of the service routine.

80. (original) The method of claim 79, wherein the passed linkage return address is selected to point to a memory page having a memory attribute that raises the chosen exception on an attempt to execute an instruction from the page.

81. (original) The method of claim 79, wherein

the service routine is an interrupt service routine of an operating system for a computer architecture other than the architecture native to the computer;

the service routine is invoked by an asynchronous interrupt; and

the caller is coded in the instruction set native to the architecture.

82. (previously presented, last amended 12/30/05) The method of claim 5, further comprising the step of:

without modifying a pre-existing thread scheduler of the computer, establishing an entry handler for execution at a specified entry point or on a specified entry condition to the thread scheduler, the entry handler programmed to save a context of an interrupted thread and modify the thread context before delivering the modified thread context to the thread scheduler.

83. (previously presented) The method of claim 20, further comprising the steps of:

during invocation of a service routine of the operating system, passing a linkage return address to the service routine at which to resume execution on completion of the service, the linkage return address being deliberately chosen so that an attempt to execute an instruction from the linkage return address on return from the service routine will raise a program execution exception;

on return from the service routine, attempting to execute the instruction at the linkage return address and raising the chosen exception; and

after servicing the exception, returning control to a caller of the service routine.

84. (new) A computer comprising:

an operating system;

an interrupted thread;

an entry handler that permits execution at a specified entry point or on a specified entry condition to the operating system, said entry handler obtained without modifying said pre-existing operating system of the computer; said entry handler programmed to save a context of said interrupted thread and modify the thread context before delivering the modified thread context to the operating system; and

an exit handler that permits execution on resumption from the operating system following an entry through the entry handler, said exit handler obtained without modifying the operating system;

the exit handler programmed to restore the context saved by a corresponding execution of the entry handler.